A SERIAL BINARY MULTIPLIER

The present invention relates to a serial binary multiplier for performing fixed point multiplication in data processing apparatus.

Central processing units of data processing apparatus generally incorporate a multiplier unit for performing multiplication operations. Typically such multiplier units are based on well known array multiplier designs or a shift-and-add algorithm. Multiplier units of this kind are generally optimised for performance (i.e. processing power and speed) or for compact implementation.

One example of a multiplier unit having compact size is described in our co-pending international patent application No. GB97/01520.

However, the performance of a serial multiplier, unlike an array multiplier design, is dependent on the total transmission delay in performing a sequence of operations as the serial data is received. The total transmission delay is a combination of several delays in the sequential operation of the multiplication process, namely: a delay as the data is routed to the input of the multiplier; a delay as the data passes through the interconnect; and the multiplier operation delay.

In monolithic design circuit performance has improved many fold as semiconductor processing techniques have lead to smaller and smaller component geometries. Contemporary integrated circuit process technology enables the manufacture of deep sub-micron circuit elements with physical dimensions of less than one micron. The performance of these circuits is often no longer determined by the operation of the active circuit components but is dominated by the interconnect delay between them.

The difference between the performance of active components, for example transistors, and the interconnect, or routing, is greatly exaggerated in the implementation of programmable circuits such as Field Programmable Gate Arrays (FPGAs), where greater flexibility in the interconnect structures adds further to the delay imposed on signals passing through them.

An alternative known approach to constructing a high performance multiplier is to base the design around a look-up-table. This is demonstrated in Altera's FLEX

10K device. Using this technique all the possible results of the multiplication process are stored in a table and the input operands are used to choose one result from the table. The size of such multipliers becomes very large when, say, operands of 8 bits or more are used. The multiplication of n-bit wide operands requires a table with 2^{2n} entries. An improvement to this design is to use multiple smaller look-up tables followed by a calculation step. This technique is also shown in Altera's FLEX 10K device. The latter technique reduces the size of the multiplier but degrades the performance since a further calculation step is required after a preliminary result has been selected from the look-up table.

INSA7 US-A-5539685 describes the conventional process of multiplication by repeated shift and add operations. An improved high-speed multiplier device is then described. The improved device requires both the operands to be loaded into separate registers before the shift and addition process is commenced. The latter process is first completed before the partial process product is stored in a temporary register. Since each process is dependent on the previous process being complete, the multiplication operation is still relatively time consuming.

In Computer Design (Pennwell Publ. –US ISSN 00104566) Volume No. 5, May 1972, pages 115-121, XP-002130443 an article entitled "2's Complement Anthimetic Operations" by S.Sklar describes the technique of Booth Coding for 2's complement multiplication. This technique provides a multiplication process with performance improvements. An arithmetic operation is performed on one operand by reference to decoded bits of a second operand. The required arithmetic operation is determined by the results of the decoding process.

JP03116327 describes a high speed multiplier. Multiples of a first operand are pre-calculated and made available at a selector. The selection of the appropriate operand multiple is made according to the current pair of bits of the second operand. The selection requires the presence of both current bits of the second operand and only then can the addition operation be performed.

It is an object of the present invention to obviate or mitigate the aforesaid disadvantages and to improve the performance of the data processing function of multiplication.

According to a first aspect of the present invention there is provided a serial binary multiplier for multiplying two binary operands to provide a final product, the multiplier comprising means for storing at least one first operand, a register for storing a partial product of the multiplication operation, means for receiving elements of a second operand serially, a calculation unit for calculating all possible results being the sum of the partial product and the product of the first operand with all possible values of the element of the second operand, said possible results being calculated during transmission of the second operand, means for selecting either one of the possible results or the currently stored partial product on the basis of the value of the received element of the second operand, means for shifting the partial product in the register to provide a new partial product, and means to output the contents of the register as the final product when all bits of the second operand have been received.

By using the calculation unit to calculate partial products whilst the second operand is transmitted the delay in transmitting the data is less significant in the overall time required to conduct the multiplication process.

Preferably the second operand comprises a plurality of elements each comprising an m-bit word. In an embodiment where m=1 the calculation unit is an adder.

The calculation unit calculates all possible results on the basis of the value of the first operand and the value of previously received elements of the second operand.

The means to output the contents of the register preferably provides the final result in serial form.

In one preferred embodiment the first and second operands and the final product are in two's complement form and the possible results are calculated from the first operand, the partial product and the previously received bit of the second operand. In such an embodiment the calculation unit is an adder and subtractor and may take the form of a single circuit capable of addition and subtraction, the operation being determined by the value of the previously received bit.

According to a second aspect of the present invention there is provided a method of operating a serial binary multiplier for multiplying two binary operands to provide a product comprising the steps of storing a first operand, storing a partial product in a register, transmitting elements of a second operand serially whilst simultaneously calculating all possible results being the sum of the partial product and the product of the first operand with all possible values of the element of the second operand, selecting either one of the possible results or the currently stored partial product on the basis of the value of the received element of the second operand, shifting the partial product in the register to provide a new partial product, and outputting the contents of the register as the final product when all bits of the second operand have been received.

Specific embodiments of the present invention will now be described, by way of example only, with reference to the accompanying drawings in which:

Figure 1 is a block diagram of an embodiment of an m-bit binary serial multiplier according to a first embodiment of the present invention;

Figure 2 is a block diagram of an embodiment of a 1-bit binary multiplier according to a second embodiment of the present invention;

Figure 3 is a block diagram of the multiplier of figure 2 adapted for two's complement operation according to a third embodiment of the present invention;

Figure 4 is a table showing the calculation process of the multiplier shown in figure 3; and

Figure 5 is a timing diagram for a single cycle of the multiplier operation.

Referring now to the drawings, figure 1 shows the structure of an m-bit serial multiplier which performs the multiplication operation on a locally stored first operand B and a second operand A that is transmitted to multiplier in the form of a serial stream of m-bit wide data elements, the m bits of each data element being received in parallel and multiple serial data elements forming the complete operand data word.

The multiplication process is performed by a calculation unit that comprises a bank of 2^m registers 1 and a bank of 2^m adders 2. The registers 1 store all possible 2^m results of multiplying the first operand B with all possible (2^m) values of the second operand A. Each register 1 within the bank stores one result of multiplying the first operand B with an assumed value of the second operand A. Each of these 2^m multiplication results is passed to one of the adders 2 in the bank of 2^m adders where it is summed with a partial product of the overall multiplication process that is stored in a shift register 3. The results of the addition process are then passed to a multiplexer 4.

A decoder (not shown) receives the m-bit serial input data element of the second operand A and on the basis of this, selects the appropriate correct result via the multiplexer 4. Thus the input data is used to select a pre-calculated result late in the calculation process. The selected (partial) result is then stored in the shift register 3 which reformats the partial result by shifting the stored data by m-bits to the right. The partial result is then recirculated to the input of the bank of adders 2. The multiplication process described above is then repeated for the next received data element of the second operand A until the whole of the input data word of the second operand A has been received and processed. If the input data represents the value zero then the recirculated output of the shift register can simply added to the register 3 rather than selecting the appropriate adder output. The final result in the shift register 3 is transmitted to a parallel to m-bit serial converter (not shown) which outputs the final result (product) in the original m-bit serial format.

The above described multiplier allows the parallel operation of both the multiplier operation (including addition of the products to the partial result in the shift register 3) and the input data transmission. Using a locally stored first operand B a number of possible multiplier results is pre-calculated independently of the second operand A and added to the partial result from the previous cycle. In this way the multiplication process delay and the data transmission delay occur simultaneously, or in parallel. The second operand A is only needed to complete the multiplication process by selecting one of the pre-calculated results. By employing a decoder that selects the appropriate partial result the delay generally associated with the multiplication process is reduced, whilst avoiding the need for a large look-up table of possible results.

It will be appreciated that by using the locally stored first operand B in the preliminary multiplication process, the number of possible pre-calculated results is greatly reduced in comparison to conventional multipliers based on look-up table designs.

Figure 2 shows an embodiment of the present invention that is used to multiply 1-bit serial input data. A 1-bit serial multiplier is highly suited to realisation within a programmable device, since implementing programmable interconnects between functional units that only require a single point-to-point connection is both practical and well known.

The operation of the 1-bit multiplier is similar to that of the generic m-bit multiplier example described earlier, however, using a 1-bit wide input format allows a novel optimisation of the circuit.

Since the input data of the second operand must be either a 1 or a 0 then only one dynamic calculation is required as there are only 2 possible results, one of which is a null operation (i.e. multiplication by zero). The structure of the 1-bit multiplier varies from the m-bit multiplier in that the calculation unit only comprises a single register to store the first operand B and a single adder. Parts corresponding to those of figure 1 are indicated by the same reference numerals increased by 100 and are not further described except insofar as they differ from their counterparts in figure 1.

The calculation unit 1, 2 shown in Figure 1 can be constructed in the 1-bit multiplier embodiment of figure 2 by using a register store for operand B and a single adder 102. The implementation of such a circuit is well known. When the multiplication operation is initiated the previous serial input bit is taken to be a zero. Once the current signal data input bit of the second operand A has been received it is used to determine whether the selected result is to be the result dynamically calculated by the adder (the sum of the received bit of the second operand A and the partial product in the register 103) or the previous partial result (i.e. no operation is performed). The final result is output via a parallel to serial converter 105.

Figure 3 shows a multiplier design for multiplication of 1-bit operands in two's complement format. The serially transmitted second operand A is decoded by a decoder 207 and the output provides instructions to an adder/subtractor 208 to choose the dynamic calculation operation i.e. either to add or to subtract the local operand B to or from the partial result that is fed back from the shift register 203. These add and subtract instructions are decoded from the previous signal data input bit and allow the dynamic calculation to be performed in parallel with the current signal data bit being transmitted. When the multiplication operation is initiated the previous serial input bit is taken to be a zero. Once the current signal data bit has been received and decoded it is used to determine whether the selected result is to be the result dynamically calculated by the multiplier or the previous partial result (i.e. no operation is performed) according to the table shown in figure 4.

The timing diagram for a single cycle of the 1-bit two's complement multiplier operation is shown in Figure 5. The opening part of the clock cycle is available for the independent dynamic calculation of partial result(s) on the basis of the previously received data bit, and for the transmission of the current data bit. This is shown as "Tmult" in Figure 4. The remaining part of the clock cycle is then dedicated to the late select process that requires simple decoding of the current serial input data bit, which may be easily constructed with simple logic gates to give very high performance. The delay attributed to this process is shown as "Tselect" in Figure 4. Clearly, overlapping the data transmission delay and the multiplier operation delay in this late select

multiplier design offers greatly improved performance over traditional serial multipliers.

It will be appreciated that numerous modifications to the above described design may be made without departing from the scope of the invention as defined in the appended claims. For example, the shifting of the partial product stored in the shift register 3, 103, 203 may be performed by any equivalent operation such as modifying the connections to the register. The term "shifting" is used in the claims with the intention of incorporating such equivalent operations.